NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

The algorithms in the GMPLS Lightwave Agile Switching Simulator (GLASS)

Version: Draft 1.0

TABLE OF CONTENTS

1 INTRODUCTION

The goal of GLASS is to simulate a network and help in the validation of protocols and algorithms. The algorithms play an important role in the efficiency of a network. GLASS provides an interface to make easy the implementation and the simulation of algorithms. The first part presents the structure of the algorithm interfaces, and the second part shows the tools that have been created to help the implementation of a new algorithm.

2 ARCHITECTURE

The algorithms in GLASS are entities that can be accessible from any node in the network. They also have the possibility to access any information of the network components. Currently the execution of an algorithm does not take time in the simulation. Research is done to include a computation delay that will be configurable by the user.

GLASS uses the same interface for all the algorithm but considers three types of algorithms:

- Routing algorithms: Algorithms that are of type Routing compute one or more routes between a source and a destination according to a quality of service.
- Wavelength algorithms: Their goal is to create a lightpath along a specific route to carry information depending on a quality of service (bandwidth, maximum delay…).
- Routing and wavelength algorithms (RWA): These algorithms are computed the route and the wavelength at the same time.

This chapter explains in details the implementation of the algorithms.

2.1 CLASS DESCRIPTION

Figure 1: UML diagram of algorithms

The package **gov.nist.antd.optical.algorithm** contains the basic classes for algorithm. It is composed of three classes:

The class **AlgorithmException** is an exception that is thrown during the execution of an algorithm when an error occurs.

The class **AlgorithmContainer** represents a global container for a specific network that stores the list of available algorithms. A static method allows any entity to retrieve an algorithm and to call its execution.

The interface **Algorithm** contains the generic methods that are necessary for configuring and calling the execution of algorithm. This interface provides two types of execute methods. On one hand, the parameters are of type Vector and on the other hand of type Array. This is to be more flexible is the implementation of algorithms and on the protocols that will call the algorithms. The method *execute(…)* is called to request a route or a path on the **OpticalConnection** [1] objects given as parameter.

In addition to this, the package **gov.nist.antd.merlin.algorithm** contains the class **AlgorithmTemplate**. This class has been created to make easier and faster the implementation of a new algorithm. By providing the basic configuration, the user focuses only on the implementation of the algorithm. All the algorithms provided in GLASS extend this template. It able the user to configure the name of the algorithm and an attribute "debug" that is used to print additional debug information.

2.2 CONFIGURATION OF ALGORITHMS

By using a DML file, the configuration of the algorithms is done after the creation of the topology. The configuration of the algorithm must be done in the section "AlgorithmContainer" of the DML file.

The minimum configuration requires the name of the algorithm, name that will be used during the simulation, and the class name to use.

Example:

```
AlgorithmContainer [
  algorithm [
    name ShortestPathDistance
    use gov.nist.antd.merlin.algorithm.route.shortestpath.distance.ShortestPathDistance
    debug true
  ]
  algorithm [
    name BestFit
    use gov.nist.antd.merlin.algorithm.wavelength.bestfit.BestFit
  ]
]
```

Figure 2: Example of DML configuration

This example will create two entries in the *AlgorithmContainer*. One for an algorithm called **ShortestPathDistance** and one for **BestFit** using the specified class.

By using the GLASS Topology and Simulation Creator (GLASS-TSC), the user can add, remove, and modify the algorithms.

3 HOW TO CREATE A NEW ALGORITHM

There are two parts that must be considered when creating an algorithm in GLASS:

- The configuration of the algorithm and
- The running of the algorithm.

In GLASS the terminology "execute an algorithm" means that we request a route or a path on specific connection object. If an algorithm needs to compute the backup routes of all the links for protection, then we don't execute the algorithm but we "configure" it. The difference is that the configuration is done before the simulation runs.

3.1 ROUTING ALGORITHM

This section explains what are the important steps to create a routing algorithm.

The routing algorithm has for objective to compute one or more possible routes for an **OpticalConnection**.

3.1.1 THE PARAMETERS

When a routing algorithm is executed, one of the execute(…) methods will be executed. The only difference is the way of accessing the elements.

The first parameter contains one or more OpticalConnection instances that need to have a route.

Figure 3 shows the implementation of the class OpticalConnection.

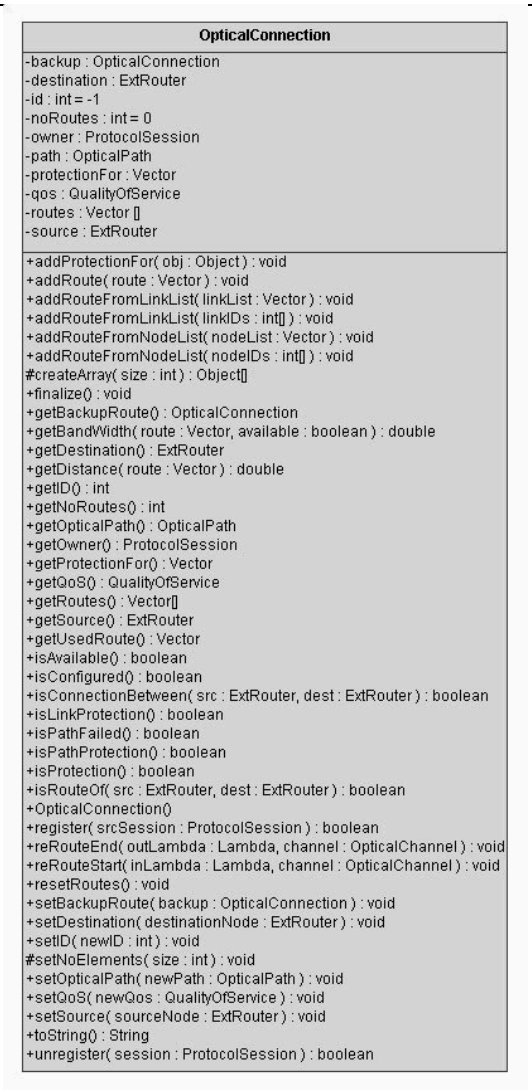

Figure 3: UML diagram of the class OpticalConnection

The **OpticalConnection** objects given as parameter may already have a route. In this case, it is more secure to call the method *resetRoutes()* of the object to clean the previous results (but not the path if existing). It is up to the resources management to clean the paths before they are recomputed.
The constraints of the route are located in the **QualityofService** located in the connection object, like the source and destination node.

The return value of the algorithm is the Vector or Array of the routes given in parameter.

3.1.2 EXECUTION

During the execution, the algorithm may find an error. In this case, it should throw an **AlgorithmException**. If no route can be found, this is not consider as an error and the algorithm must leave the route object as it is without any possible routes.

When the algorithm finds a route, it must add it in the list of possible routes of the **OpticalConnection**. The internal structure of the possible routes is an array of Vector. Each Vector contains one or more **PtPBundle** [1]. The algorithm can follow the internal structure but GLASS provides easy methods to convert lists of links or nodes into the internal structure. These methods are integrated to the **OpticalConnection** and also in the utility classes [3].

The return value of a routing algorithm must be the list of OpticalRoutes, whether a route has been found or not.

The way the algorithm uses the framework is free to the developer and in the case of the algorithms provided in GLASS, we create a graph out of the net and then run a regular algorithm (for example ShortestPathDistance). See the package gov.nist.antd.merlin.protocol.routing.util for more information [3].

3.2 WAVELENGTH ALGORITHM

The wavelength algorithm is the second step for the creation of the connection. Once the routing algorithm has computed list of possible routes, the wavelength algorithm tries to compute the lightpath.

3.2.1 THE PARAMETERS

The parameters of the wavelength assignment are the same has for the routing algorithm. The list of OpticalConnection contains the connections that need a lightpath.

3.2.2 EXECUTION

The wavelength algorithm should make sure that the connection has already some possible routes available. If there is no possible route in an OpticalConnection then the wavelength should not do anything. It also needs to check if there is already a path in this given connection then it must free the resources used. To do so, it can call the *resetSwitches ()* of the optical path.

The wavelength algorithm does not need to care about the setting of the switches along the new path. The caller of the algorithm should do it. This allows instantaneous set-up of the lightpath or dynamic set-up [1].

The algorithm must create a path according to a quality of service. The algorithm must be concerned about the capabilities of the switches along the path. Some may not have wavelength conversion. Some links may also not have enough bandwidth.

If no path can be found, the algorithm does not have to modify the **OpticalConnection** because a route without path is consider as a failed connection.

4 DML SCHEMAS OF EXISTING ALGORITHMS

algorithm [
  name ShortestPathDistance
  use
gov.nist.antd.merlin.algorithm.route.shortestpath.
distance.ShortestPathDistance
  debug %S1
]
	Configuration for the ShortestPathDistance routing algorithm.
	Attribute debug is used for extra information (default = false).

algorithm [
  name ShortestPathDistanceSRLG
  use
gov.nist.antd.merlin.algorithm.route.shortestpath.
srlg.ShortestPathSRLG
  debug $S1
]
	Configuration for the ShortestPathSRLG routing algorithm.
	Attribute debug is used for extra information (default = false).

algorithm [
  name KShortestPath
  use
gov.nist.antd.merlin.algorithm.route.shortestpath.
k.KspDisjoint
  debug $S1
  k $I1
]
	Configuration for the KshortestPathDistance algorithm.
	Attribute debug is used for extra information (default = false).
	Attribute k indicates the number of possible routes must be searched (default =1).

algorithm [
  name BestFit
  use
gov.nist.antd.merlin.algorithm.wavelength.bestfit.
BestFit
  debug $S1
]
	Configuration for the BestFit wavelength assignment algorithm.
	Attribute debug is used for extra information (default = false).

5 REFERENCES

[1] Connection, Route and Path in GLASS

By NIST/ANTD

[2] GMPLS Lightwave Agile Switching Simulator – Topology and Simulation Creator (GLASS-TSC)

By NIST/ANTD

[3] Utils in the GMPLS Lightwave Agile Switching simulator

By NIST/ANTD